Global Multi-Threaded Instruction Scheduling: Technique and Initial Results

Guilherme Ottoni

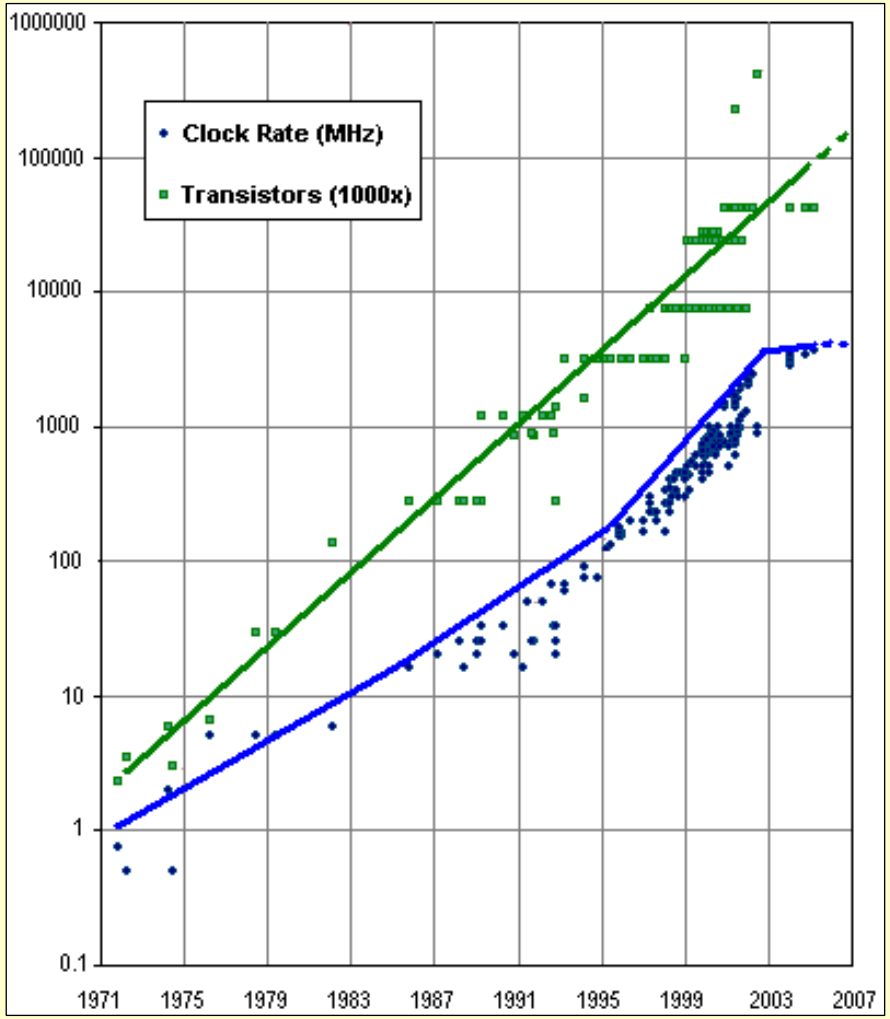
David I. August

Liberty Research Group

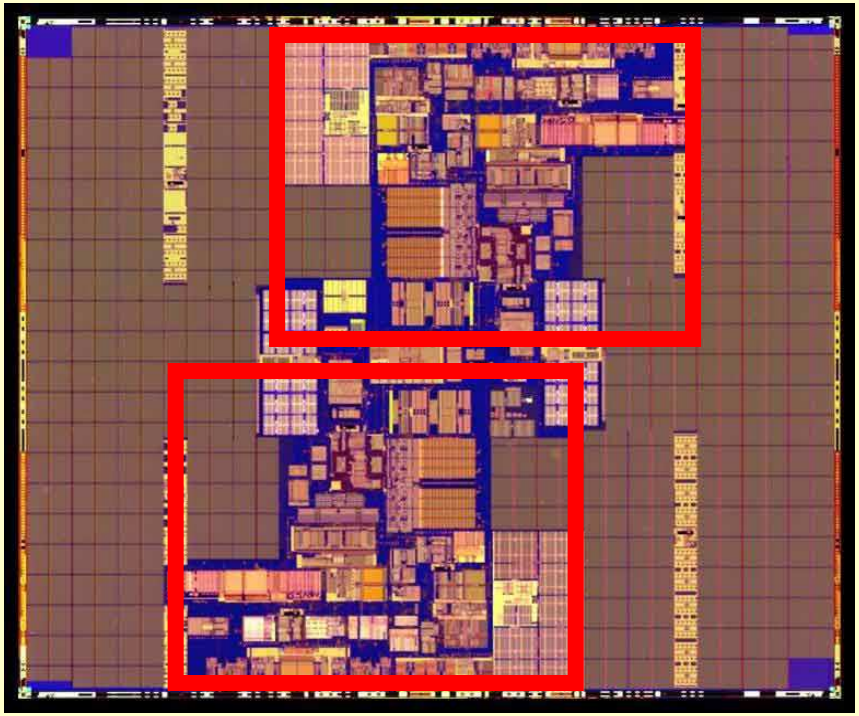
Department of Computer Science

Princeton University

A Fundamental Change…

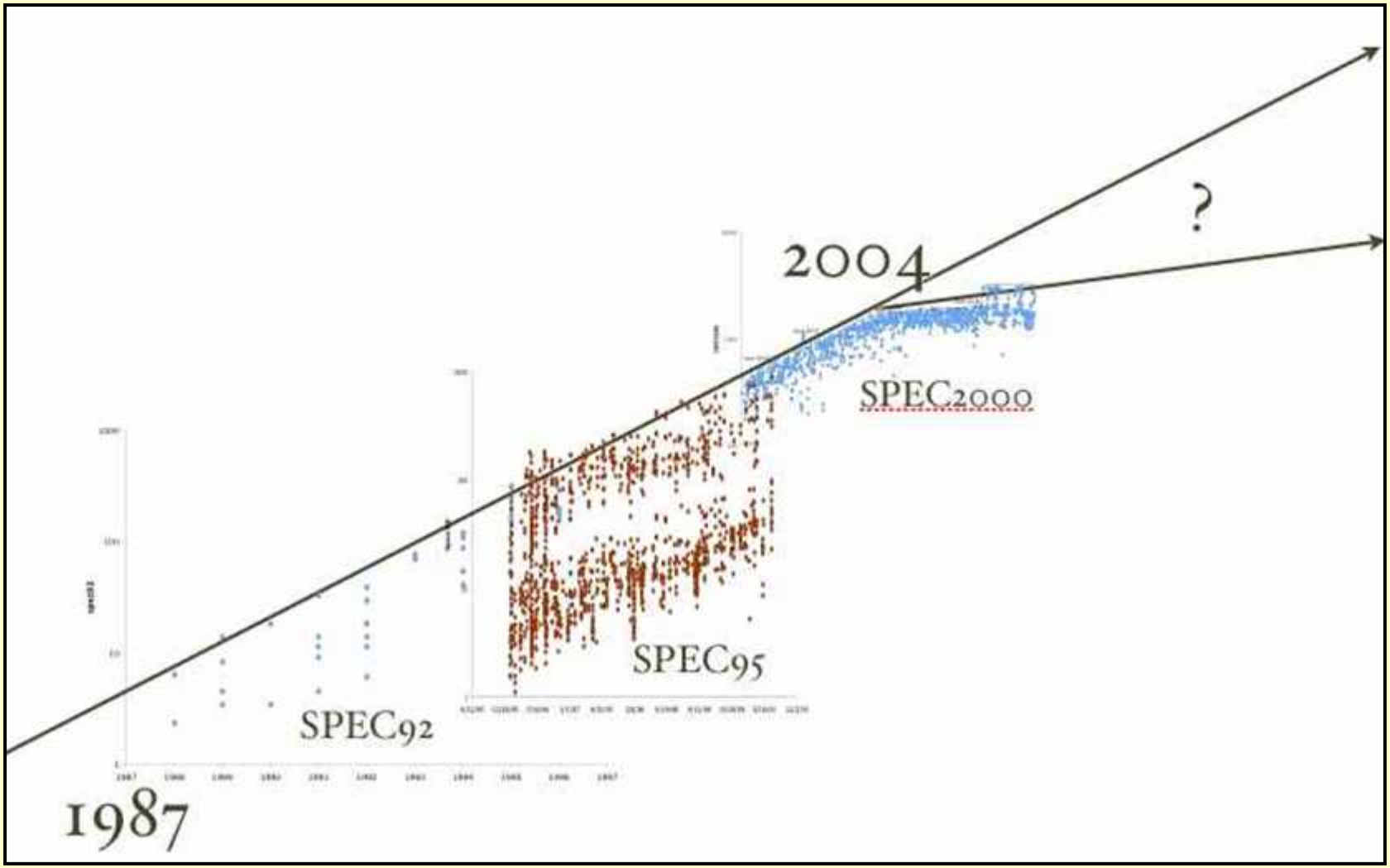


Source: Intel, Wikipedia, Sutter/Dr. Dobbs Journal



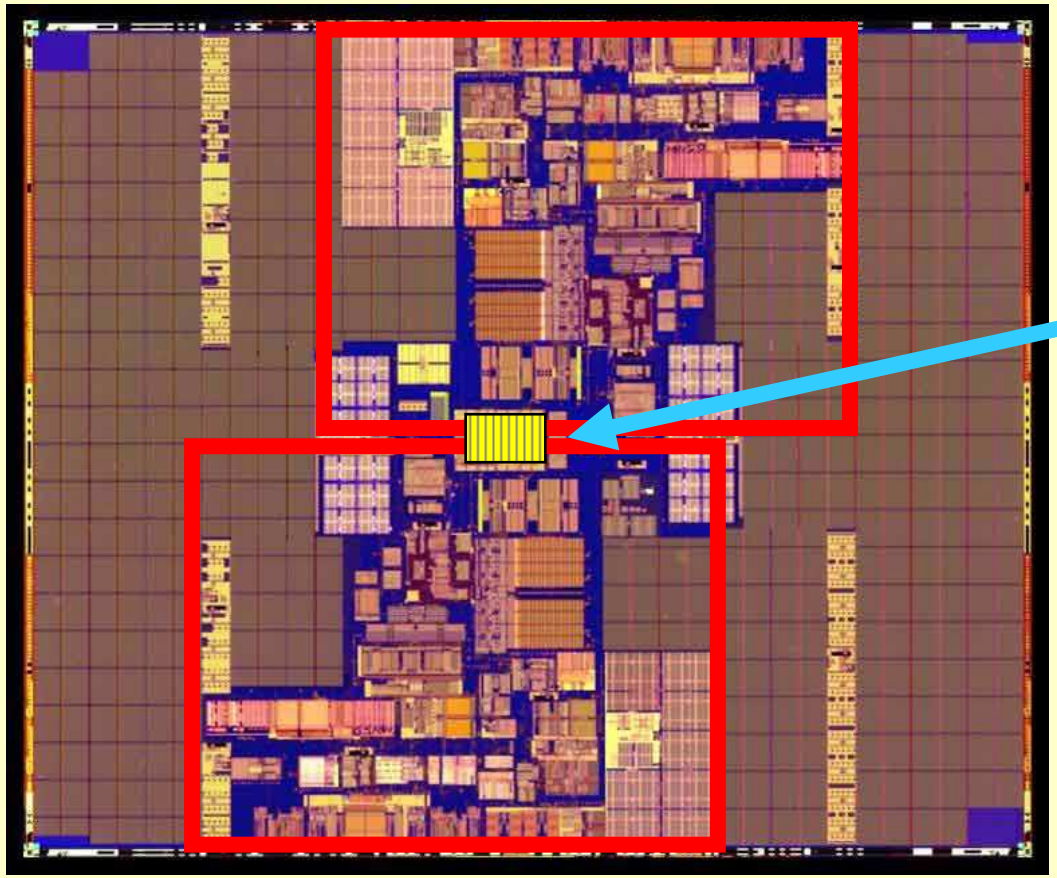
Intel Montecito: Source Intel

The Biggest Problem in Computer Architecture in Decades…



Graph Courtesy Mark Oskin

Thread-Level Parallelism for CMPs



- Benefit from multi-core specific opportunities
  - High bandwidth
  - Fine-grained comm./sync. mechanisms
- Enables more coupled TLP

Compiler Scheduling for MT Processors

```
s1 = s2 = 0;
for (p = head; p != NULL; p = p->next)
    s1 += p->value;
for (i = 0; a[i] != 0; i++)
   s2 += a[i];
printf("%d\n", s1 * s1 / s2);
```

- Short BBs
- Very little ILP within basic blocks
- No opportunities for Local MT scheduling [Lee'98,'02]

Need to perform **global** MT scheduling!

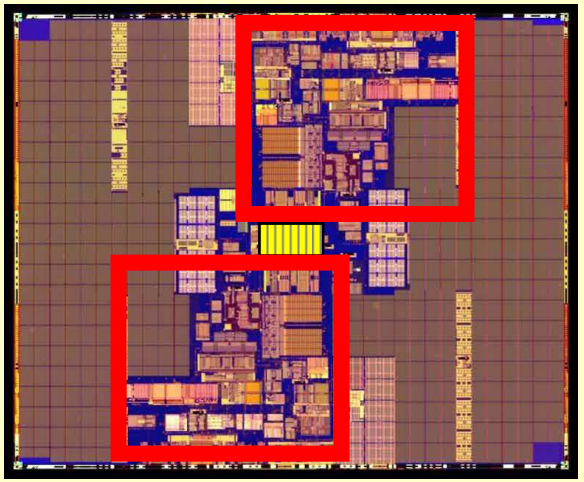
```
B1:   move    r1 = 0
      move    r2 = 0
      load    r3 = [head]
B2:   branch  r3 == 0, B4
B3:   load    r4 = [r3]
      add     r1= r1, r4
      load    r3 = [r3+4]
      jump    B2
B4:   move    r5 = @a
B5:   load    r6 = [r5]
      branch  r6 == 0, B7
B6:   add     r2 = r2, r6
      add     r5 = r5, 4
      jump    B5
B7:   mult    r7 = r1, r1
      div     r8 = r7, r2
```

Issues in Global MT Instruction Scheduling

	Local MT [Lee'98,'02]	**Global MT**
CFG	1 basic block	Arbitrary
Instructions	Always execute	May not execute
Instr. Profile Wgt.	Same	May be very different
Dependence Graph	Acyclic	Cyclic
Dep. Types	Only Data	Data + Control
Dep. Occurrence	Always exercised	May not occur

Compared to single-thread scheduling:

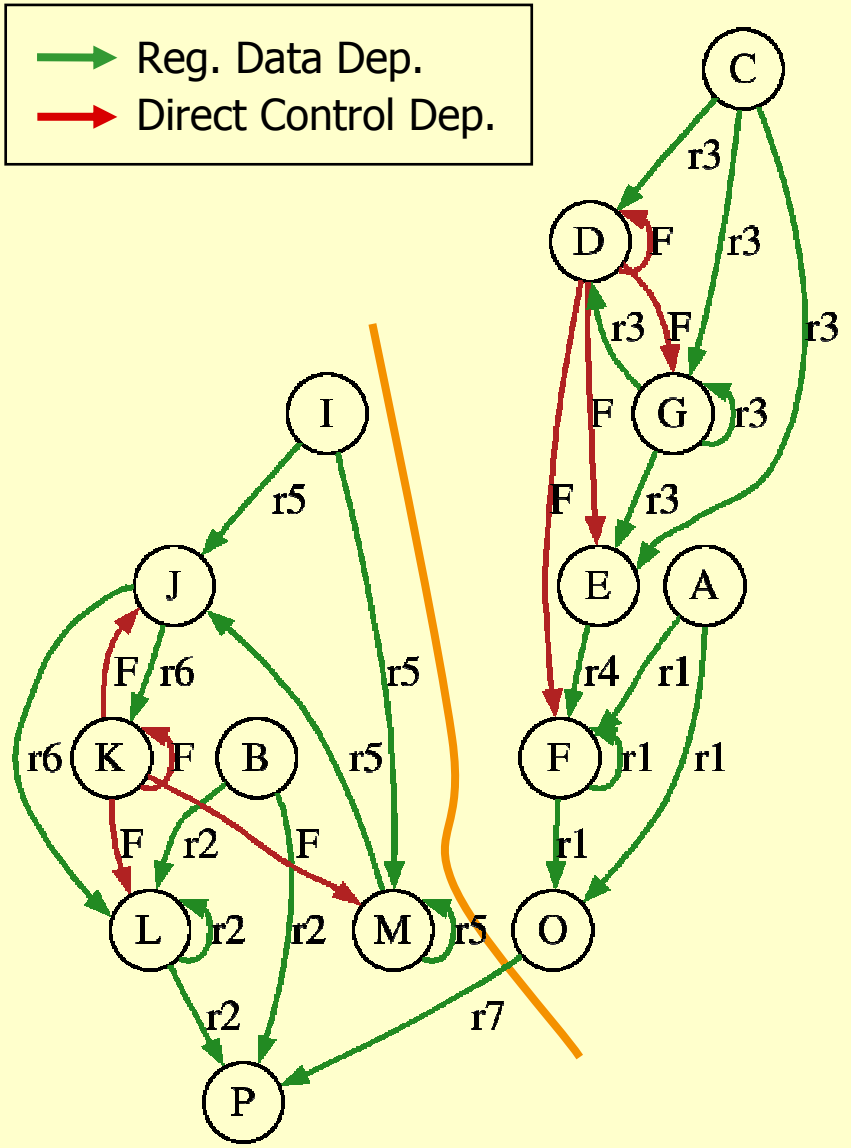
- Control units: multiple, 1 per core
- Communication cost: inter-core latency > 0, extra ops



Global MT Scheduling Approach: Overview

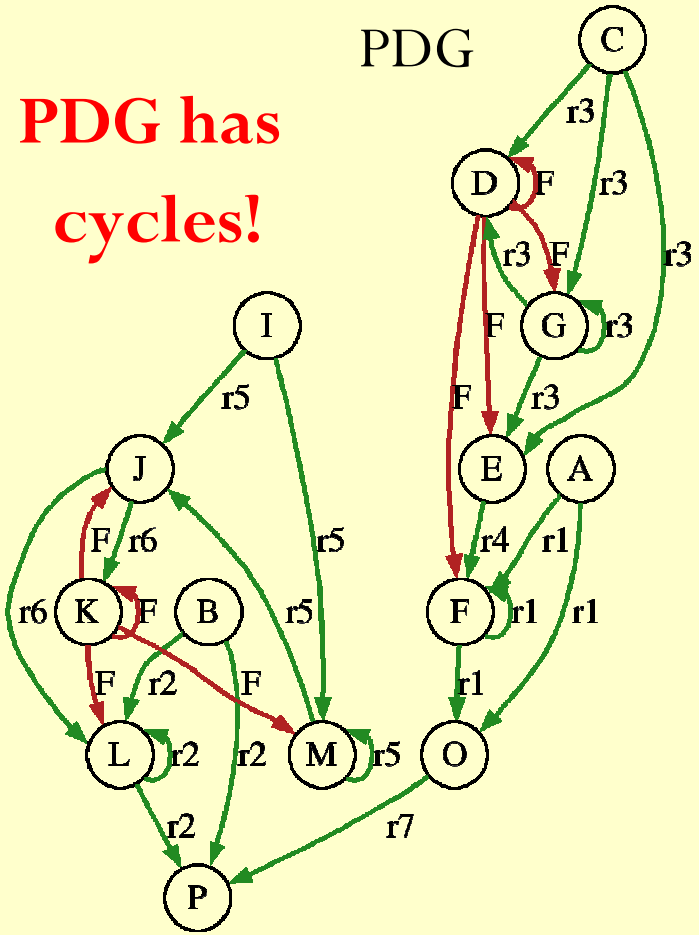
1. Program Dependence Graph
2. Thread Partitioning
3. MT Code Generation

```
(A)  B1:  move    r1 = 0
(B)       move    r2 = 0
(C)       load    r3 = [head]
(D)  B2:  branch  r3 == 0, B4
(E)  B3:  load    r4 = [r3]
(F)       add     r1 = r1, r4
(G)       load    r3 = [r3+4]
(H)       jump    B2
(I)  B4:  move    r5 = @a
(J)  B5:  load    r6 = [r5]
(K)       branch  r6 == 0, B7
(L)  B6:  add     r2 = r2, r6
(M)       add     r5 = r5, 4
(N)       jump    B5
(O)  B7:  mult    r7 = r1, r1
(P)       div     r8 = r7, r2
```



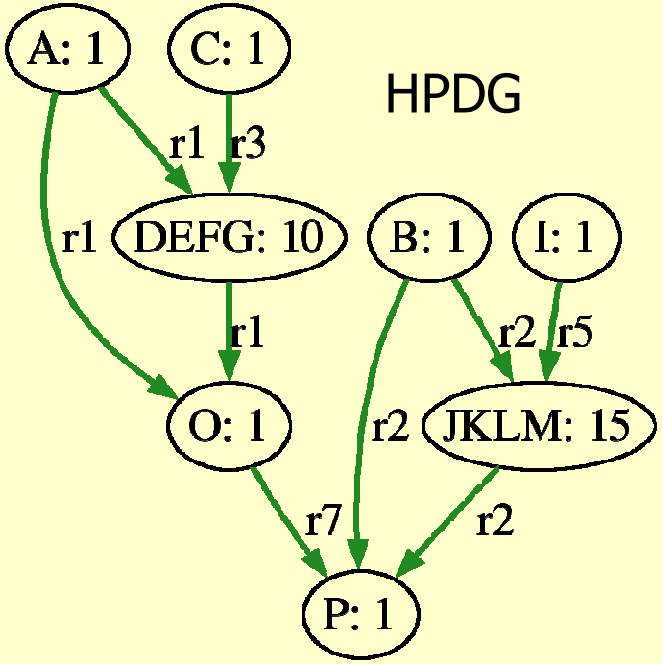
MT Code Partitioning

- Conflicting goals:
  - Maximize parallelism
  - Minimize communication



Reduce to acyclic graph (HPDG):

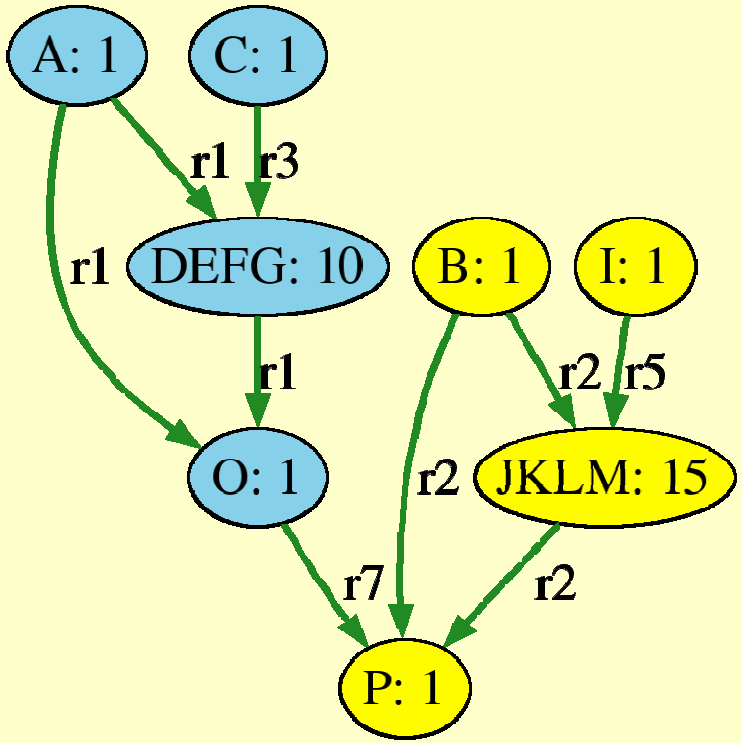
1. Coalesce inner loops
2. Ignore loop-carried dependences



MT Code Partitioning: Accounting for Comm. Costs

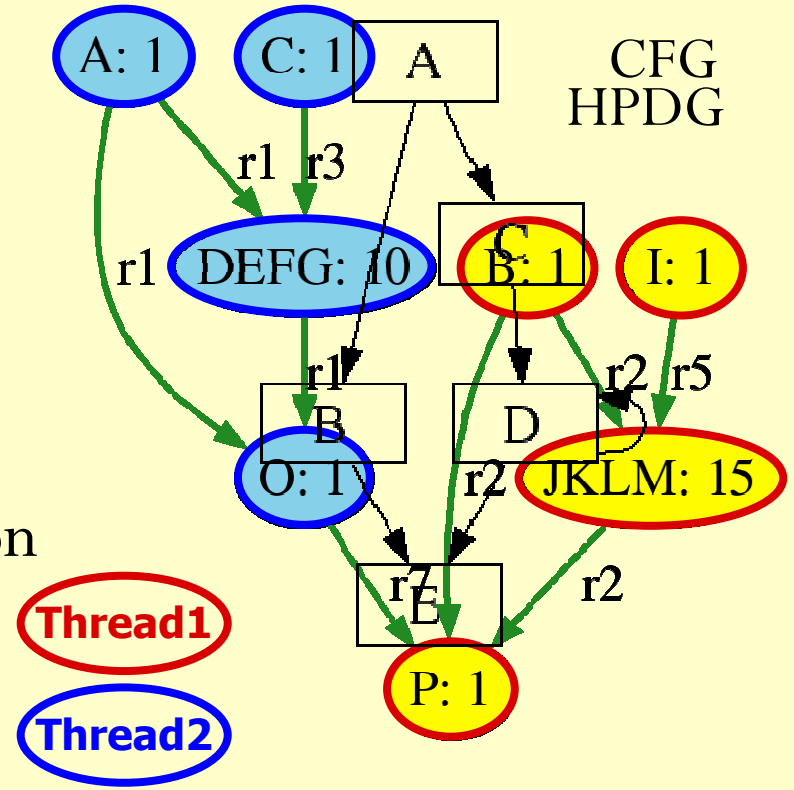
- Use a pre-scheduling clustering pass
  - Dominant Sequence Clustering (DSC) [Yang+Gerasoulis'94]
- Nodes in the same cluster are scheduled on the same thread

HPDG



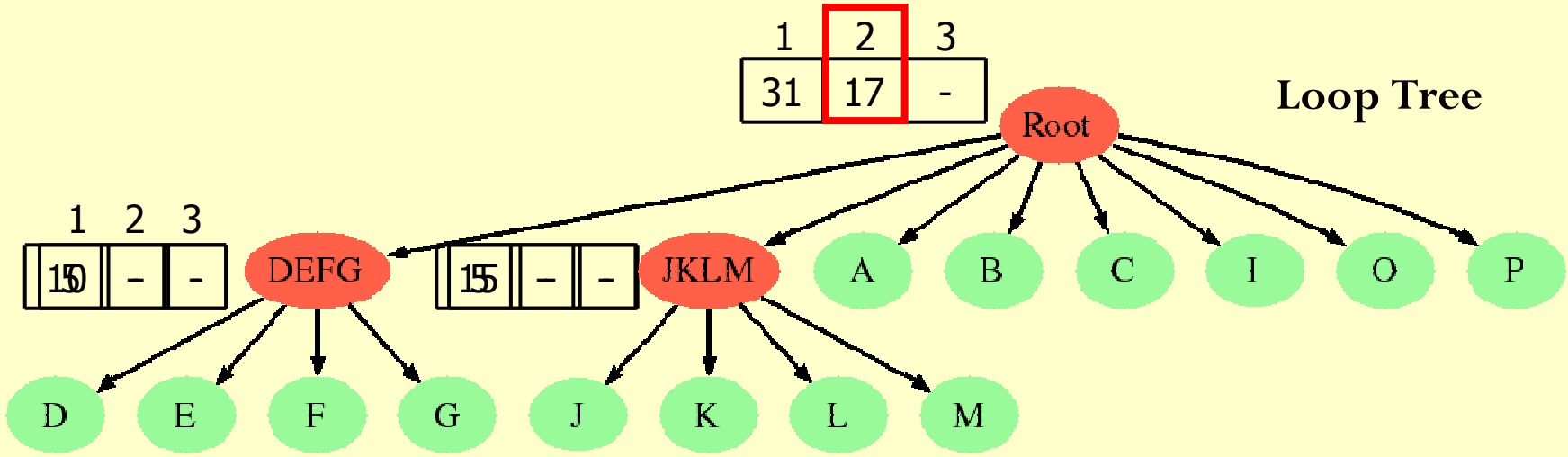
Code Partitioning: Thread-Aware List Scheduling

- Build a *virtual schedule*
- Thread-aware: different threads can simultaneously execute instructions with different control conditions
- Control Relations:
  - Control Equivalent
  - Mutually Control Exclusive
  - Control Conflicting
- Thread for cluster is chosen when scheduling its first node
- ↑Parallelism X ↓Communication
  - Startup overhead
  - Communication overhead
  - Resource-conflict overhead



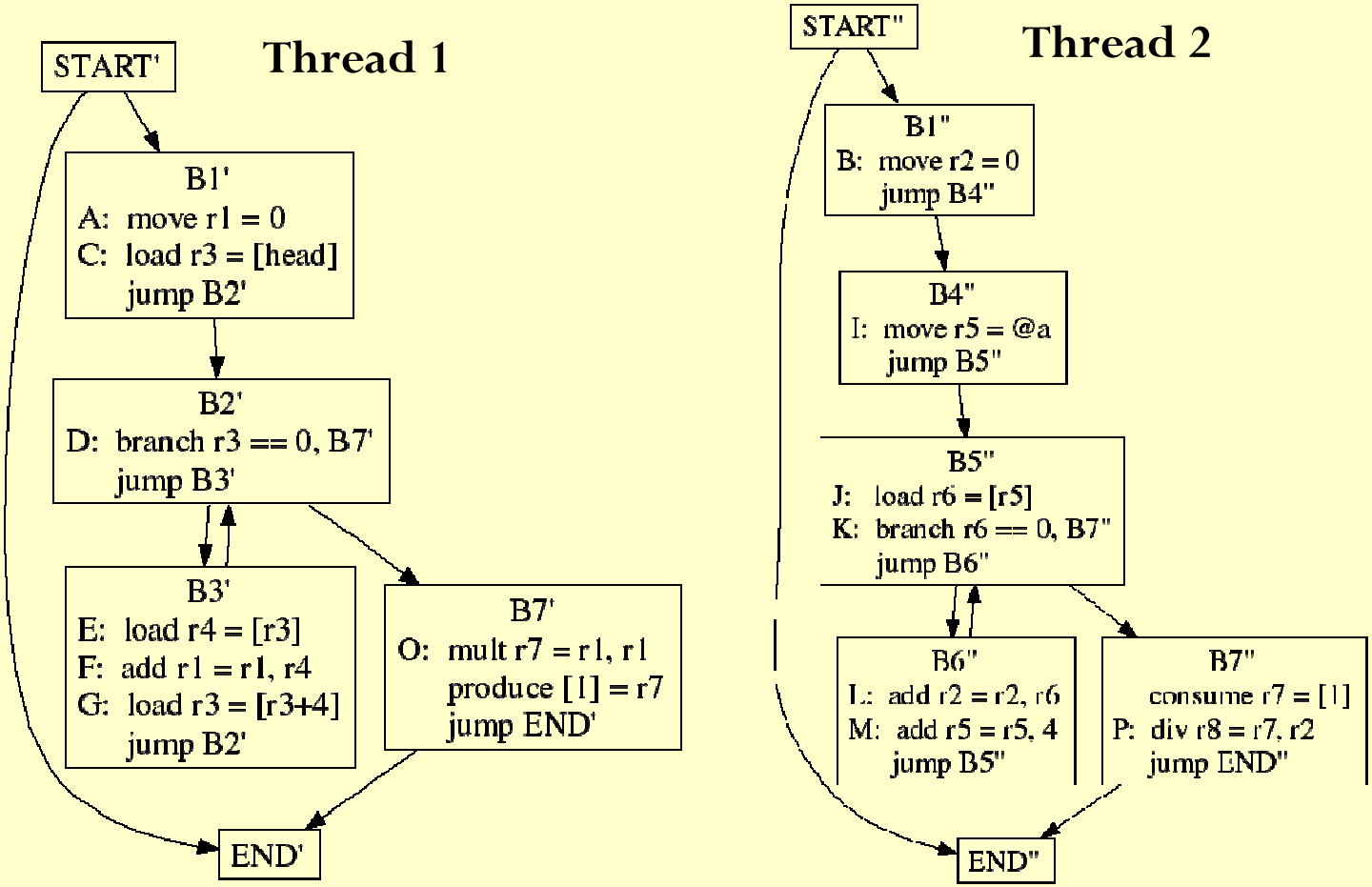
MT Code Partitioning: Handling Loop Nests

- Loop nodes are **not** scheduled solely based on clustering
- Region is hierarchically scheduled bottom-up using loop tree
- At each node, use list scheduling to compute latency on 1..N threads
- When scheduling inner loop L_i, choose # of threads k that minimizes:
  $\text{latency}_{Li}(k) + \text{cycle\_available}(k)$
- Pick best # of threads for root, and corresp. MT partition



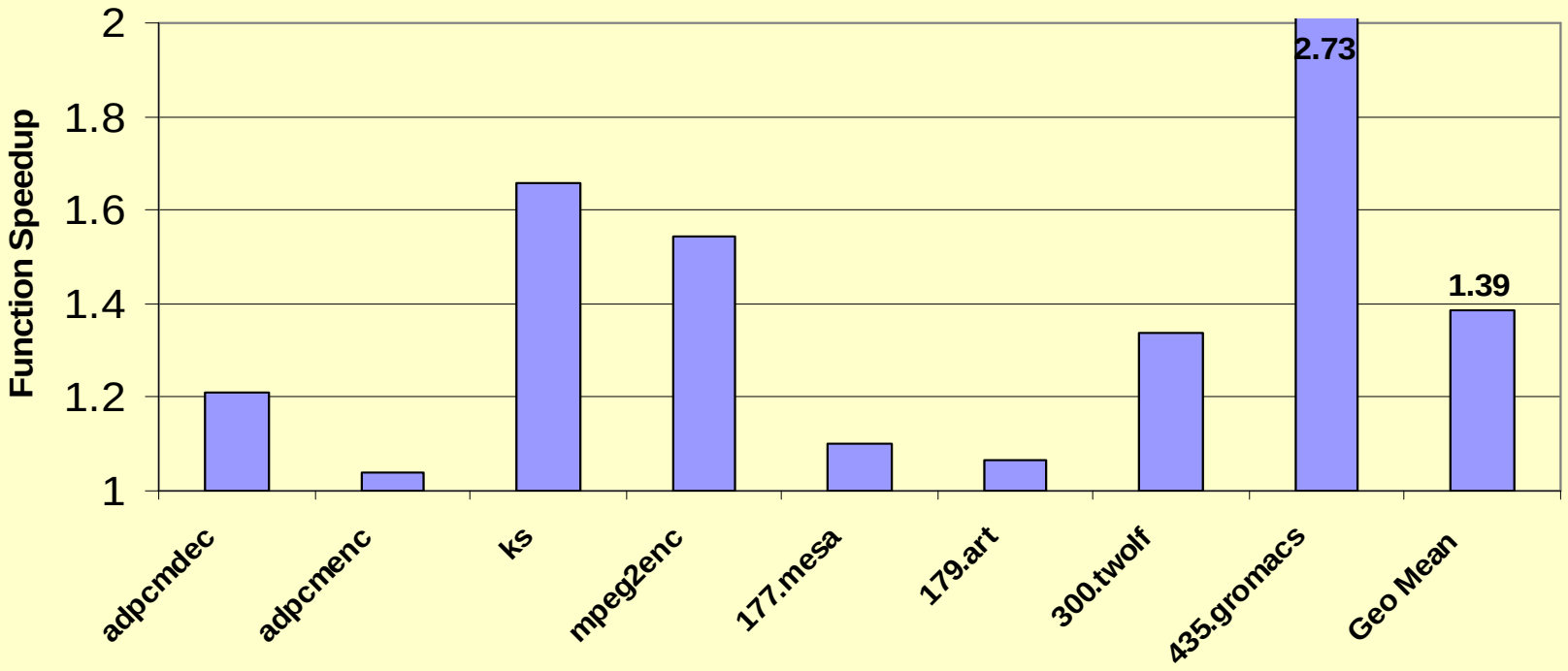
MT Code Generation

- Based on PDG and original CFG
- Preserves all PDG dependences, thus semantics [Sarkar'92]
- Inserts produce/consume instructions to satisfy inter-thread dependences



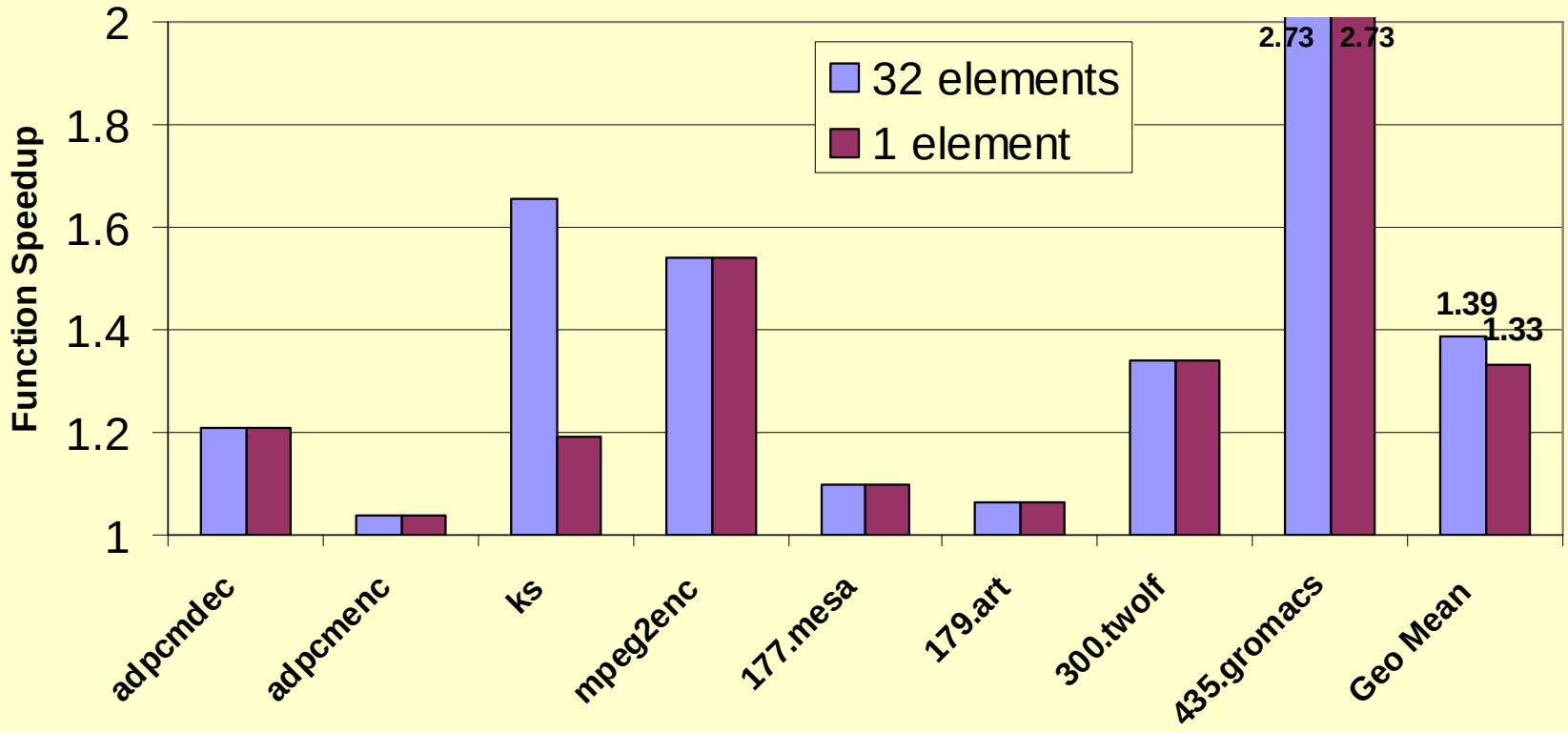
Evaluation

- GMTIS implemented in the back-end of Velocity compiler [PLDI'06]
- Accurate dual-core Itanium 2 model [MoBS'05]
- Synchronization Array support for comm./sync. [PACT'04]
  - ISA extended with produce/consume instructions
  - 2-cycle inter-core communication latency
- Important application functions selected (30-100% total execution)



Evaluation: Smaller Communication Queues

- Base model: 32-element queues
- Modified model: reduced to 1 element
- Only ks is affected



Conclusion

- Multi-cores expose novel parallelization opportunities
- Global multi-threaded instruction scheduling can effectively exploit TLP opportunities
  - Applicable to a number of applications
  - Benefit from additional resources (control/func. units, registers)
  - Alternative to power-hungry ILP techniques
- Need fine-grained communication/synchronization support
- Future research directions:
  - More exploration of scheduling heuristics
  - Combination with other TLP techniques, e.g. DSWP

EXTRA SLIDES

Evaluation: Larger Communication Latency

- Base model: 2 cycles of inter-thread communication latency
- Modified model: increased to 10 cycles